Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Karin Piškur

Popravljanje vejic v slovenskih besedilih z orodjem LanguageTool

diplomsko delo

visokošolski strokovni študijski program prve stopnje računalništvo in informatika

Mentor: izr. prof. dr. Marko Robnik Šikonja

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:
Popravljanje vejic v slovenskih besedilih z orodjem LanguageTool

Tematika naloge:

Raba vejice je eden od jezikovnih elementov, pri katerem v slovenskem jeziku nastane največ napak. Orodje LanguageTool nam omogoča, da besedilo, ki ga pišemo, sproti pregledujemo in predlagamo popravke. Na podlagi analize korpusa Lektor, ki je izpostavil problem postavljanja vejic v slovenščini, sestavite pravila, ki bodo v okolju LanguageTool prepoznavala in opozarjala na najpogostejše napake pri postavljanju vejic. Sestavljena pravila in predlagane izboljšave ustrezno ovrednotite.

Izjava o avtorstvu diplomskega dela

Spodaj podpisana Karin Piškur sem avtorica diplomskega dela z naslovom:

Popravljanje vejic v slovenskih besedilih z orodjem LanguageTool

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelala samostojno pod mentorstvom izr. prof. dr. Marka Robnika Šikonje,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 5. septembra 2015 Podpis avtorja:

Predvsem bi se rada zahvalila dr. Damjanu Popiču za vso pomoč in spodbudo pri nastajanju diplomskega dela. S pomočjo vašega usmerjanja in svetovanja nisem samo uspešno dokončala diplomskega dela, ampak še bolj vzljubila slovenski jezik. Najlepša hvala!

Zahvaljujem se mentorju izr. prof. dr. Marku Robniku Šikonji za vodenje in nasvete, ki so pripomogli k strokovnosti mojega dela.

Zahvaljujem se svoji družini, predvsem za potrpežljivost ter za to, da so mi vedno stali ob strani, me vzpodbujali in verjeli vame.

Želela bi se zahvaliti prijateljici Mateji, ki je del moje družine in brez katere ne bi bila to kar sem. Hvala, ker si!

Hvala tudi ostalim prijateljem in vsem, ki so mi kadarkoli stali ob strani.

Kazalo

Povzetek

Cilj diplomskega dela je dodati pravila za pisanje vejice v orodje LanguageTool. S pomočjo analize korpusa Lektor smo raziskali, katera pravila za postavljanje vejic v slovenščini piscem povzročajo največ težav. Glede na dobljene rezultate smo v Slovenskem pravopisu 2001 podrobneje raziskali pravila za pisanje vejice pred vezniki *in*, *ali* ter *da*. Po končani raziskavi smo poskusili implementirati čim več pravil za odprtokodno orodje LanguageTool, ki ga lahko uporabljamo kot samostojen program, spletni vmesnik ali v odprtokodnih pisarniških paketih LibreOffice in OpenOffice. Nekaj pravil smo uspešno implementirali, vendar ne vseh, saj bi za to potrebovali označevalnik besedila, ki ga orodje LanguageTool za slovenski jezik še ne vsebuje. Na koncu smo opravili še evalvacijo pravil, s katero smo preverili njihovo pravilnost, uporabnost in uporabniško izkušnjo.

Ključne besede: LanguageTool, slovenski jezik, vejica, ločila, korpus Lektor.

Abstract

The aim of the thesis is to add the rules for comma usage to the LanguageTool program. Using the Lektor corpus, we examined which rules for comma usage are causing the most issues in written Slovene. In view of these results, we analyzed the rules for comma usage before conjunctions *and*, *or* and *that* in Slovenian ortography 2001. After finishing the analysis, we tried to implement comma placement rules for the open source program LanguageTool, which can be used as a stand-alone desktop application, as web interface or in open source office suites LibreOffice and OpenOffice. Some of the rules were successfully implemented. For all of the rules to be implemented, we would need the part-of-speech tagger, which is not a part of the LanguageTool for Slovene, yet. We evaluated the rules, taking their accuracy, applicability and user experience into account.

Keywords: LanguageTool, Slovenian language, comma, punctuation mark, Lektor corpus.

Poglavje 1

Uvod

Kot kažejo najnovejše empirične raziskave o normativnih zadregah pri pisanju v slovenskem jeziku, imajo pisci največ težav oz. največ napak napravijo pri stavi vejice [9]. Glavni razlog za to je ta, da so pravila za stavo vejice v slovenskem jeziku izrazito skladenjska, kar pomeni, da se pisci pri stavi vejice ne morejo (kaj dosti) zanašati na intuicijo, temveč morajo biti zmožni sprotne in napredne skladenjske analize, da lahko zadostijo pravopisnemu standardu. Poleg vseh drugih pravopisnih pravil so pravila za stavo vejice zbrana v Slovenskem pravopisu 2001 [12], načeloma pa je stava vejice odvisna od treh komponent: pomena, logike in sotvarja.

V sklopu diplomskega dela dodajamo nekaj pravil za stavo vejice v orodje LanguageTool. To orodje nam omogoča, da besedilo, ki ga pišemo, sproti pregledujemo in predlagamo popravke. Pri kasnejši analizi ugotovimo, koliko pravil za slovenski jezik vsebuje LanguageTool in zakaj je temu tako. V okviru diplomskega dela poskušamo nabor pravil v orodju dopolniti glede na najbolj akutna mesta pri stavljenju vejic v slovenščini, pri identifikaciji najbolj problematičnih mest pa se opiramo na korpusnojezikoslovno analizo korpusa Lektor.

Korpus Lektor je poleg že znanih virov izpostavil problem postavljanja vejic v slovenskem jeziku. Zato na podlagi analize tega korpusa izberemo najbolj problematične strukture pri stavi vejice. Po izboru poiščemo pravila,

ki jih najdemo v Slovenskem pravopisu 2001 [12] (v nadaljevanju SP 2001), jih analiziramo ter jih poskusimo implementirati v okolju LanguageTool, tako da bodo prepoznavala in opozarjala na najpogostejše napake pri postavljanju vejic.

Nova implementirana pravila je treba ovrednotiti, saj testni primeri, ki jih uporabimo pri implementaciji, ne zagotavljajo pravilnosti in uporabnosti pravila. To storimo s testiranjem le-teh na konkretnih besedilih in nato primerjamo, kako uspešni smo bili pri odkrivanju napak. Tako ugotovimo, ali je bilo odkritje napačne stave vejice dovolj relevantno, da lahko napisano pravilo uporabimo, ali pa ga je treba dopolniti. Iz rezultatov lahko izvemo tudi, ali so naša pravila uporabna in kakšna je uporabniška izkušnja tako s pravili kot s samim orodjem LanguageTool.

V poglavju 2 spoznamo jezikovno ozadje našega dela, v katerem podrobneje predstavimo korpus Lektor in rezultate analize tega korpusa. Nato raziščemo pravila za stavo vejice pred izbranimi vezniškimi strukturami. Temu sklopu sledi poglavje 3, ki je posvečeno orodju LanguageTool in v katerem opišemo njegovo strukturo in delovanje. Sledi implementacija pravil, v primeru, da določeno pravilo ni bilo implementirano, pa je podana utemeljitev. V poglavju 4 se posvetimo evalvaciji implementiranih pravil. V zadnjem, 5. poglavju predstavimo sklepe in izpostavimo segmente, ki jih nismo uspeli implementirati.

Poglavje 2

Jezikovno ozadje

Kodifikacija slovenskega jezika je v razmerju do stave vejic kompleksna, saj je za pravilno stavo vejice potrebno poglobljeno skladenjsko znanje in zmožnost jezikovne analize, to pa predvsem zato, ker smo slovenski sistem ločil v veliki meri prevzeli iz nemško govorečega okolja. Kompleksnost pravil pri uporabnikih povzroča številne zadrege. To sta ugotovila tudi Popič in Logar v prispevku *Med dvema ognjema: kje stoji vejica v slovenskih gimnazijah?* [5]. Z anketno raziskavo sta ugotovila, da se težave pogosto pojavljajo (tudi) v srednješolskem izobraževanju, čeprav bi ravno tam morale biti odpravljene oz. je ravno v tem segmentu izobraževanja standardizacija izjemnega pomena. Ugotovljeno je bilo, da se večina dijakov pri postavljanju vejic ravna tako po občutku kot samih pravilih, kar pa v številnih primerih ne zadostuje za pravilno stavo vejice, saj je ta pogosto odvisna od formalno-logičnih pravil. Tovrstna raba je denimo v izrazitem nasprotju s pragmatičnimi načeli stavljenja vejice, ki so značilni za angloameriško okolje, v katerem vejica temelji predvsem na pomenu in segmentaciji.

Stava vejice je za slovensko kulturno okolje izjemnega pomena, saj velja za preizkusni kamen jezikovne kultiviranosti. Poznavanje pravopisa je tudi nasploh, zaradi slovenske zgodovine, zaznamovane z življenjem v podrejenem političnem položaju, bistveno bolj pomembno kot v kulturah, ki so bile v zgodovini v boljšem položaju pri uveljavljanju svojih jezikovnih pravic, podobno

ugotavljata tudi Popič in Fišer v prispevku *Vejica je mrtva, živela vejica* [4]. Analiza tvitov, za katere velja, da se v njih uporablja nestandarden jezik, je pokazala, da je v le-teh vejica enakomerno in pogosto postavljena. Vendar to še zdaleč ne pomeni, da pisci tvitov nimajo težav s stavo vejice, ravno nasprotno, vidimo lahko, da napake/težave niso vedno povezane s formalnostjo besedila, ampak, kot smo že prej omenili, s samimi pravili v SP 2001 [12]. Za našo temo je pomembno predvsem to, da so težave pri rabi vejice relativno predvidljive, tudi na leksikalni ravni, saj imamo v slovenščini t. i. atraktorje, pri katerih je napačna stava vejice še posebno pogosta. Popič in Fišer tako analizirata problematična mesta, kjer se v stavi pojavijo besede *kljub*, *glede* in *zaradi*.

V tem poglavju se posvečamo vprašanju, pred katerimi jezikovnimi elementi se v slovenskem jeziku najpogosteje pojavljajo težave s postavljanjem vejice. Raziščemo, v katerih strukturah vejica največkrat umanjka in v katerih je največkrat vejica odveč.

Najprej predstavimo korpus Lektor, katerega podatke uporabimo tudi za analizo. S pomočjo korpusnojezikoslovne analize poskušamo ugotoviti, na katerih mestih imajo pišoči v slovenščini največ težav.

Na podlagi rezultatov analize korpusa se odločimo, katerim skladenjskim položajem se posvetimo in podrobneje predstavili pravila za stavo vejice, ki jih podaja SP 2001 [12].

2.1 Korpus Lektor

V tem razdelku predstavimo korpus Lektor [2], obenem pa podamo tudi splošnejše informacije o korpusih in njihovi zgradbi.

2.1.1 Besedilni korpusi

V nadaljevanju povzamemo članek Tomaža Erjavca z naslovom *Računalniške zbirke besedil* [3]. Besedilni korpusi zajemajo besedila v naravnem jeziku, ki so pridobljena iz množičnih medijev, knjižne produkcije, stripov, interneta,

reklamnih besedil, navodil priloženih izdelkom, prepisov parlamentarnih razprav ipd. Ta besedila so skupaj z označbami strukturirano shranjena v digitalni obliki. Besedilni korpusi so uporabni za jezikoslovne raziskave in strojno učenje, vendar le takrat ko so le ti dovolj veliki in urejeni.

Uporabljene oznake so leme (osnovne oblike besed), oblikoskladenjske oznake, skladenjske oznake in še nekatere druge oznake, ki se ne uporabljajo le za namene korpusnega jezikoslovja, ampak tudi za namene prepoznavanje govora in strojnega prevajanja v računalniški lingvistiki.

Korpuse glede na jezik delimo na enojezične in večjezične. Enojezični korpusi vsebujejo besedila v enem jeziku, večjezični pa v več jezikih. Poznamo dve vrsti večjezičnih korpusov; primerjalne in vzporedne. Primerjalni so namenjeni predvsem za kontrastivne (»ki pojave enega jezika proučuje glede nasprotja, primerjave z drugim jezikom, protistaven« [11]) študije in vsebujejo primerljiva besedila v različnih jezikih. Primerljiva so lahko tematsko, jezikovnozvrstno, besedilnovrstno ... Vzporedni večjezični korpusi pa so namenjeni predvsem prevodoslovju. Vsebujejo poravnano izhodiščno besedilo in prevod v enega ali več jezikov, kar nam omogoča, da vidimo način prevajanja besedila. Poznamo še referenčne korpuse, ki so osnovna zvrst korpusov in služijo kot jezikovni standard, govorjene in govorne korpuse, ki vsebujejo govor oz. transkripcijo govora, korpuse podjezikov, ki vsebujejo jezik z nekega področja, vzorčne korpuse, ki vsebujejo le fragmente besedil in spremljevalne korpuse, ki so dinamični in se spreminjajo skupaj s spreminjanjem/posodabljanjem jezika.

Korpus Lektor

Korpus Lektor je nastal v okviru doktorske disertacije dr. Damjana Popiča na Filozofski fakulteti, na Univerzi v Ljubljani [9], iz katere v nadaljevanju povzamemo podrobnosti o njem. Korpus Lektor je enojezični korpus prevodov in avtorskih besedil, katerih avtorji oz. prevajalci imajo najmanj 7. stopnjo izobrazbe. To pomeni, da če določena slovnična in pravopisna pravila predstavljajo težave njim, lahko sklepamo, da jih najverjetneje predstavljajo

tudi večini ljudi, ki pišejo v slovenskem jeziku. Vsa vključena besedila so novejša oz. mlajša od deset let. Korpus je prosto dostopen, tudi z namenom, da ga lahko uporabijo raziskovalci.

Korpus je oblikovan v formatu XML (eXtended Markup Language), ki velja za standard pri oblikovanju korpusov. Vsebuje metapodatke o lektorju, avtorju in publikaciji besedila. Za korpus so pripravili tudi oznake za označevanje popravkov. Nadkategorije teh oznak, katere vsebujejo različno število kategorij, so:

- slog,
- oblika,
- pravopis,
- skladnja in
- pragmatika.

V korpusu Lektor je omogočeno klasično iskanje, iskanje po lastnostih besedil in lektorjev, možnost pregledovanja korpusa po izbrisanem besedilu, po vstavljenem besedilu in po tipu napak, kar lahko vidimo na sliki 2.1 (npr. kot oznako izberemo napake *P-LociloStava*).

Da bi bil korpus čim bolj reprezentativen, so vključena besedila, ki so jih lektorirali različni lektorji, segment besedil, ki so prevodna, pa zajema prevode iz različnih jezikov. Skupno korpus Lektor tako zajema okoli milijon besed iz (poljudno)znanstvenih besedil, tematika besedil pa je različna; vendarle pa gre zgolj za neliterarna besedila, tj. besedila, pri katerih slog ne igra pomembne vloge.

Namen korpusa Lektor je celostna analiza lektorskih popravkov v prevodnih in avtorskih besedilih v slovenskem jeziku. Za objektivno spremljanje jezikovnih popravkov v besedilih v digitalni obliki je avtor disertacije uporabil korpusnojezikoslovno analizo. S tovrstno analizo je mogoče zajeti celoten proces od koncepta do končne oblike besedila.

Slika 2.1: Iskanje napak v korpusu Lektor.

V korpusu lahko najdemo dve različici besedil, izvorno avtorsko besedilo in lektorirano besedilo. Besedila so v obliki XML sestavljena tako, da lahko prikažemo obe besedili hkrati. Lektorski posegi so določeni s tipi po vnaprej predvideni razvrstitvi. Do korpusa Lektor lahko dostopamo v obliki XML in prek spletnega vmesnika, kjer lahko iščemo po besedilu ali metapodatkih. Poleg glavnega namena korpusa, ki je objektiven vpogled v procese lektoriranja besedil, lahko z njim spremljamo tudi najpogostejše jezikovne težave pišočih v slovenskem jeziku.

Ker so bili najpogostejši popravki pravopisni popravki povezani s stavo ločil, smo se odločili, da se posvetimo prav tem, bolj konkretno popravkoma z vstavljanjem in izbrisom vejice.

2.2 Rezultati analize korpusa Lektor

Po pregledu rezultatov za korpus Lektor in primerjavi z že pripravljenimi pravili v orodju Language Tool 2.9 (več podrobnosti o vsebini in samem orodju sledi v poglavju 3) smo ugotovili, da so nekatera pravila za postavitev vejice že pripravljena, vendar pa jih nekaj še manjka.

Glede na rezultate, pridobljene iz korpusa Lektor, nekatere strukture niso problematične. Obstaja pa nekaj besed, za katere bi bilo pravilo za manjkajočo in/ali odvečno vejico zelo potrebno. To so besede, za katere se je napaka pojavila v več kot 1 % primerov, kar lahko razberemo z grafov, ki jih je pripravil Peter Halozan, na slikah 2.2, 2.3a, 2.3b in 2.4.

Odvečne vejice:

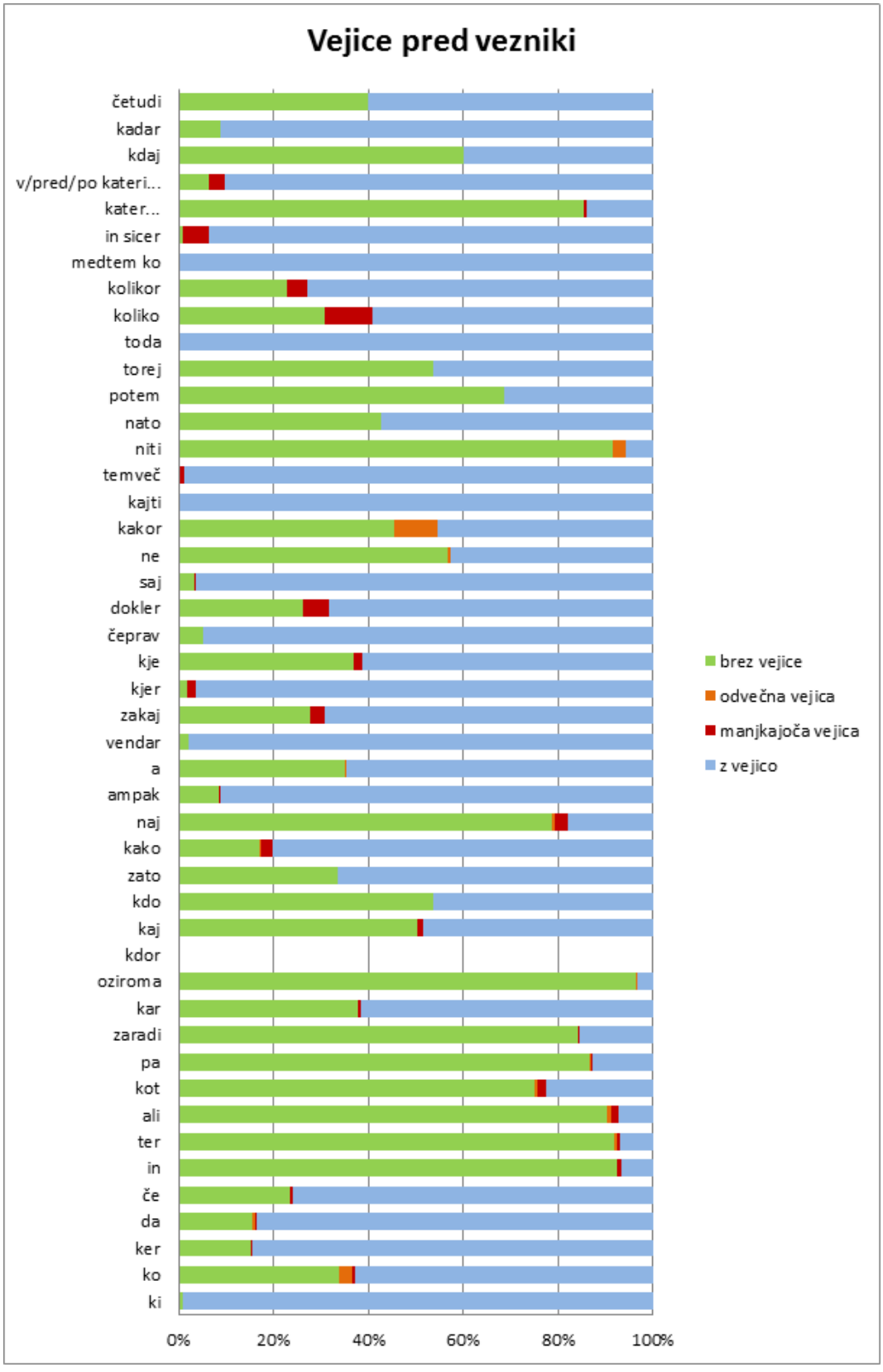
- in (9 %),
- da (8 %),
- ko (5 %),
- ali (5 %),
- kot (4 %),
- pa (3 %)

Manjkajoče vejice:

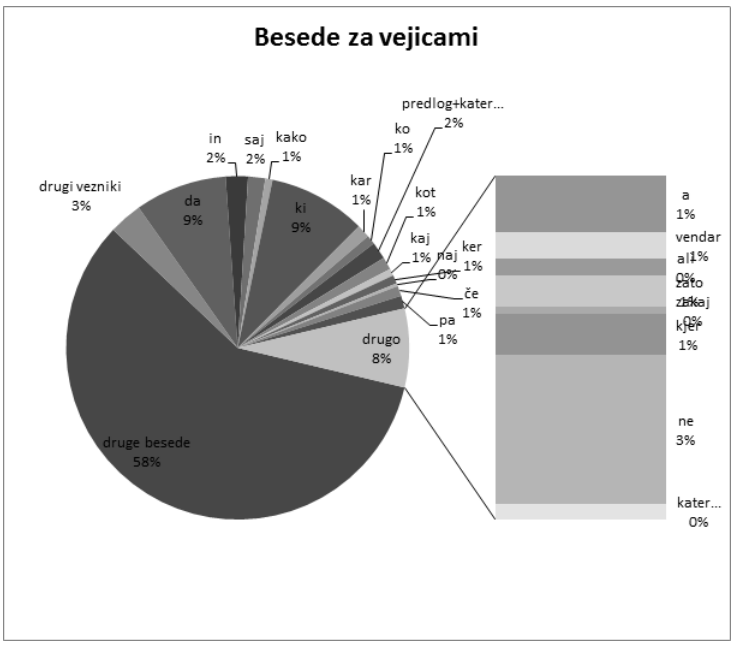
- in (18 %),
- ali (5 %),
- naj (3 %)

Posvetimo se le trem najpogostejšim, tj. primerom manjkajoče in odvečne vejice pred veznikom *in*, manjkajoče vejica pred veznikom *ali* ter odvečne vejica pred veznikom *da*.

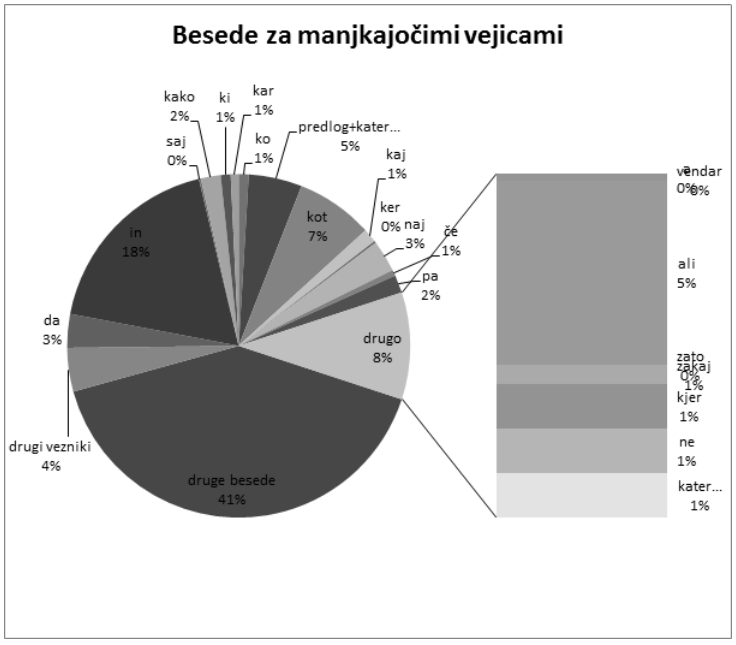
V nadaljevanju predstavimo vse tri veznike in pravila, kdaj moramo vejico pred njim pisati in kdaj ne. Pri implementaciji pravil se posvetimo temu, katera pravila so za nas pri teh veznikih relevantna ter izvedljiva in katera ne. Če namreč želimo ustrezno dopolniti orodje, moramo razumeti vsa pravila stavljenja vejice, četudi jih kasneje neposredno ne potrebujemo pri implementaciji.



Slika 2.2: Pojavitev vejice pred vezniki v korpusu Lektor.

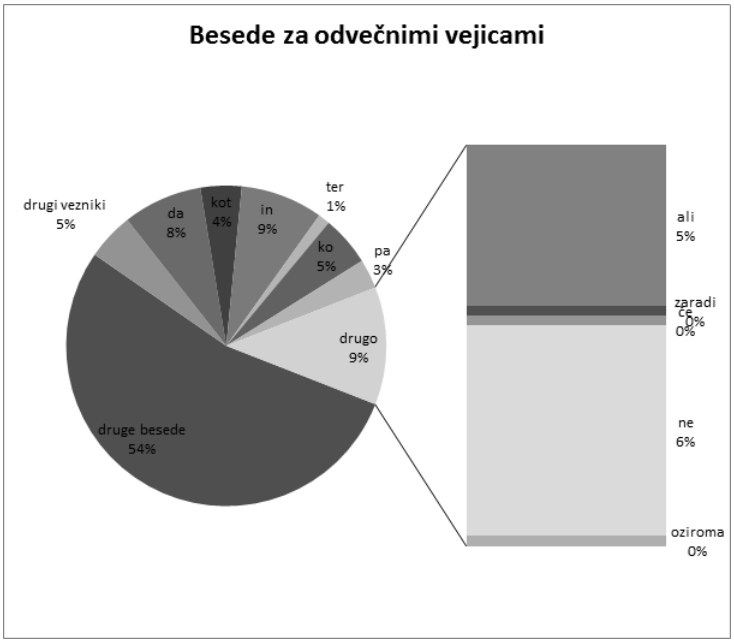


(a) Besede za vejicami



(b) Besede za manjkajočimi vejicami.

Slika 2.3: Pogostost pojavitev besed za (a) besedami za vejicami in (b) besedami za manjkajočimi vejicami v korpusu Lektor.



Slika 2.4: Pogostost pojavitev besed za odvečnimi vejicami v korpusu Lektor.

2.3 Vejica pred *in*

Nekaj primerov (ne)rabe vejice pred veznikom *in* najdemo v SP 2001 [12]. Pravzaprav se vejica ne nahaja pred veznikom *in* le v vezalnopriredni povedi.

2.3.1 Vejica pred pojasnjevalnim *in*

Veznik *in* je lahko tudi del večbesednega veznika, to sta »*in to*« ter »*in sicer*«. *In sicer* je veznik pojasnjevalnega priredja, ki ga uvaja vejica, kar je glede na analizo korpusa Lektor nepredvidljivo. V SP 2001 [12] najdemo naslednja zgleda [12] (§ 297):

- *Zbolel je za hudo, nalezljivo boleznijo (tj. za hudo, in sicer nalezljivo).*
- *Predava dvakrat na teden, in sicer v torek in petek.*

2.3.2 Vejica pred vezalnim *in*

Med skladenjsko enakovrednimi deli vejice ne pišemo. Le-te določajo tudi priredni vezniki, med katerimi je tudi vezalni *in*, zato vejice pred njim ne pišemo.

V SP 2001 [12] najdemo razlago, da vezalni *in* prepoznamo po tem, da ga ne moremo zamenjati z vezniki tipa: *vendar, zato, kajti* ipd.

Bolj preprosta prepoznava bi bila, da s pomočjo stavčne analize ugotovimo, ali sta leva in desna stran stavka in/ali povedi, katero loči veznik *in*, enakovredni ali ne. Lahko pa tudi rečemo, da vejice pred veznikom *in* ne pišemo, razen v izrednih primerih, katere predstavimo v nadaljevanju. Nekaj primerov rabe vezalnega *in* v SP 2001 [12] (§ 317):

- *Oče in mati sta mu že davno umrla.*
- *Hana prinese steklenico in kozarce ter šunko.*
- *Sem dolgo upal in se bal.*
- *Zavriskaj v lepi svet in utrgaj kresni čarni cvet in praprotnih semen nastrezi in na mah pod bukev lezi!*
- *(Premišljeval je,) kako ga ima rada in kako mu streže.*

Kot smo prej omenili za vezalni *in*, obstajajo izjeme, ko pred njim vendarle pišemo vejico, in sicer v primeru, »če jo zahteva vrinjeni ali vmesni stavek oz. dostavek« [12] (§ 321):

- *Moj prijatelj, tudi jaz imam prijatelja, in njegova žena živita v Kranju.*
- *Da pride, je rekel, in da prinese kruha.*
- *Povej mi, kje si bil ves ta čas, in pojdi potem hitro na postajo.*

in tudi »tedaj, če sta osebka prirednih stavkov slabo družljiva« [12] (§ 321):

- *Komaj je zatisnil oči, se je zbudil, in dan je bil.*

2.3.3 Vejica med nadrednim in odvisnim stavkom

SP 2001 [12] pravi: »Ta vejica se stavi ne glede na to, ali je nadredni stavek pred odvisnim, za njim ali pa ga obdaja« [12] (§ 325):

- *Dejal sem jim, da ga ni doma, in jih odslovil.*
- *Povej mi, s kom občuješ, in povem ti, kdo si.*

Vendar pa ima tudi to pravilo posebnost: »Kadar prideta skupaj priredni in podredni veznik pred vmesnim odvisnikom, pišemo vejico samo pred prvim veznikom« [12] (§ 328):

- *Pridem sam, in če bo le mogoče, pripeljem še koga.*
- *Votlina je bila suha, in ker je ne doseže nobena sapica, tudi topla.*
- *Medveda je priklenil k stebru, in ko je spet prišel divji mož, sta se spoprijela.*

2.3.4 Vejica pred dostavkom

Iz naslednjih dveh zgledov iz SP 2001 [12] lahko vidimo, da je vejica lahko pred besedo *in*, ko je ta začetek dostavka [12] (§ 340):

- *Prinesi mi sivi plašč, in rokavice.*
- *Veronika je prišla domov, sama, in celo kmalu.*

2.4 Vejica pred veznikom *ali*

Analiza korpusa Lektor je pokazala, da raba vejice pred veznikom *ali* ni dovolj znana, saj pišoči pogosto ne vedo, kdaj vejico pred veznikom *ali* staviti in kdaj ne. Veznik *ali* tako uporabljamo kot veznik ločnega ali protivnega priredja. Lahko je tudi podredni veznik, ki uvaja pogosto vprašalne stavke in stavčne prilastke.

2.4.1 Vejica pred protivnim *ali*

Vejica stoji pred *ali*, ko le-ta nastopa v protivnem priredju. Le tega določimo tako, da dopolnjevalni stavek, ki se v našem primeru začne z veznikom *ali*, izraža neskladje oziroma nasprotje osnovnega stavka: [12] (§ 308)

- *Veliko si je prizadeval, ali uspeha ni bilo.*

2.4.2 Vejica pred ločnim *ali*

Tako kot vezalni veznik *in* tudi ločni *ali* uvrščamo med priredne veznike in tako tudi zanj velja enako pravilo nestave vejice. [12] (§ 319):

- *Ali delaj doma ali pojdi v svet.*
- *Ali ne more ali pa noče.*
- *Bo ali ne bo?*
- *(Ne vem,) ali jutri pride ali ne.*

2.4.3 Vejica pred naštevalnim *ali*

V primeru, da se naštevalni *ali* pojavi v povedi več kot dvakrat, pred vsemi njegovimi pojavitvami postavimo vejico, drugače pa je stava vejice poljubna [12]:

- *Kdo lase ti razčesava, ali mati, ali sestra, ali vila iz goščav?*

2.4.4 Vejica med nadrednim in odvisnim stavkom

»Ta vejica se stavi ne glede na to, ali je nadredni stavek pred odvisnim, za njim ali pa ga obdaja« [12] (§ 325):

- *Strah, ali se mi stvar posreči, mi je jemal spanec.*

2.5 Vejica pred veznikom *da*

Značilno za podredne veznike je, da povezujejo neenakovredne dele povedi, med njimi je tudi veznik *da*. Pred njimi vedno stoji vejica, vendar obstajajo primeri, ko vejice pred njimi ne stavimo.

2.5.1 Vejica med nadrednim in odvisnim stavkom

Vejica v tem primeru vedno stoji pred veznikom *da*, »ne glede na to, ali je nadredni stavek pred odvisnim, za njim ali pa ga obdaja« [12] (§ 325):

- *Strašna mu je bila misel, da bo tepen.*
- *Daj mu nageljček, da bo tudi on vedel, da je prvi maj.*
- *Dejal sem jim, da ga ni doma, in jih odslovil.*
- *Novica, da bomo kmalu rešeni, nam je vlila novih moči.*

Vendar pa, ko se beseda *da* v besedilu pojavi kot členek, vejice pred njo ne pišemo [12] (§ 326):

- *Ti da tega ne veš?*
- *Baje da pije.*
- *Iz Blatnega Dola da ste?*

»Vejice ne pišemo med deli večbesednega veznika« [12] (§ 329):

- *Namesto da bi se učil, je lenaril.*
- *Lenaril je, namesto da bi se učil.*
- *Kljub temu da je bil bolan, je šel na delo.*
- *To smo odločili, češ da je že čas.*

»Taki vezniški izrazi so še: tako da, toliko da, /. . . /, s tem da . . . « [12] »V nekaterih primerih razmerje med nadrednim in odvisnim stavkom lahko izrazimo tako, da prvi del prvotno večbesednega veznika prenesemo v nadredni stavek, za njim pa seveda pišemo vejico« [12] (§ 329):

- *Zeblo ga je tako, da se je ves tresel.*

2.5.2 Med pastavki in preostalim besedilom

Besedica *da* se lahko pojavi v besedilu kot samostojni členek, takrat pred njim pišemo vejico [12] (§ 346):

- *Da, vse je čisto.*
- *Tako je rečeno, da.*

Poglavje 3

Orodje LanguageTool

LanguageTool je odprtokodna programska oprema za pregledovanje črkovanja in slovnice v več kot 30 jezikih. Za bolj učinkovito pregledovanje mora orodje za vsak jezik posebej vsebovati odprtokodni POS-označevalnik, tj. slovar označenih besed. Vendar pri nekaterih jezikih še vedno ni dodan, tako kot npr. pri slovenskem jeziku. Ta označevalnik je potreben za razločevanje stavčnih členov, vendar le-tega za slovenski jezik še nimamo. Zato LanguageTool za slovenski jezik vsebuje le osnovna pravila, ki jih lahko preprosto implementiramo v jeziku XML.

V tem poglavju podrobneje predstavimo orodje LanguageTool, in sicer nekoliko podrobneje pogledamo, kako je zgrajen in kako se uporablja. Kasneje poskusimo v njem udejanjiti pravila za postavitev vejic, ki smo jih predstavili v prejšnjem poglavju.

3.1 Orodje LanguageTool

Orodje LanguageTool je odprtokodni program, ki služi slogovnemu in slovničnemu pregledovanju besedil. Program je prosto dostopen na spletni strani `https://www.languagetool.org/` in ga lahko brezplačno uporabljamo in dopolnjujemo.

3.1.1 Prvotna različica

V tem razdelku povzemamo diplomsko delo [8] Daniela Naberja na Tehnični fakulteti Univerze v Bielefeldu v Nemčiji, v sklopu katere je leta 2003 nastala prvotna različica orodja LanguageTool. Prvotno je bilo orodje napisano v programskem jeziku Python, z namenom pregledovanja slovničnih in slabših slogovnih izbir v angleških besedilih, vendar pa je prvotna različica vsebovala tudi podporo za nemški jezik. Naber je svoj program zasnoval tako, da je omogočeno enostavno dodajanje podpore za druge jezike. Nova pravila tako dodajamo v datoteko XML.

To različico je bilo mogoče vgraditi v urejevalnik besedil KWord, ki je bil del pisarniškega paketa KDE (ang. K Desk Environment, slov. K namizno okolje). Da programa ni bilo treba namestiti, je Naber razvil tudi spletno različico s preprostim spletnim vmesnikom z omejenim številom možnosti. KDE je takrat deloval le na operacijskih sistemih Linux in Unix, spletni vmesnik pa tudi na operacijskih sistemih Windows in OSX. Procesi pregledovanja v ozadju (ang. backend) so tekli dovolj hitro za sprotno uporabo pregledovalnika v KWordu. Tako so bile napake podčrtane sproti med pisanjem.

Backend in orodje za pregledovalnik v ukazni vrstici sta bila enostavna za namestitev, saj sta zahtevala samo Python brez dodatnih modulov.

LanguageTool je imel 54 slovničnih pravil, 81 parov lažnih prijateljev (»Pari besed v dveh različnih jezikih, ki imajo sicer podoben zapis ali izgovorjavo, vendar nosijo različen pomen, npr. become (slov. postati) – bekommen (slov. prejeti).« [6]), 5 slogovnih pravil in 4 pravila, ki so bila izvedena v programskem jeziku Python [8].
Pregledovalnik je bil zgrajen iz treh komponent, to so [6]:

- Ločevalnik povedi (ang. sentenceSplitter), ki je zaznal meje med povedmi in/ali stavki v vnesenem besedilu.

- Označevalnik (ang. tagger), ki je glede na statistično analizo dodelil besedam oblikoskladenjske oznake.

- Drobilec (ang. chunker), ki je besedilo z oblikoskladenjskimi oznakami »razdrobil« na besedne zveze.

Delovanje in uporaba

V tem razdelku povzamemo delovanje in uporabo orodja LanguageTool po magistrskem delu Maria Jurišića [6]. V urejevalniku besedil KWord se je pregledovanje sprožilo že s kopiranjem besedila v okno urejevalnika ali pa s pisanjem v prazen dokument. Seveda je za to morala biti vklopljena možnost samodejnega preverjanja črkovanja (ang. autospellcheck). Zanimivo je, da v prvotni različici LanguageTool ni vseboval samodejnega preverjanja črkovanja, ampak je to bilo zagotovljeno z odprtokodnim črkovalnikom ISpell, ki je bil vgrajen v KWord.

V KWordu je uporabnik razlago napake poiskal z desnim klikom na podčrtano besedo, pri čemer se je odprl meni z možnostjo LanguageTool, ki je odprla novo okno. V zgornjem delu je bila izpisana poved z domnevno napako, označeno z rdečo. V spodnjem delu je bila razlaga oziroma popravek napake. S klikom na popravek je uporabnik popravil svojo poved in s klikom na gumb OK popravke potrdil ali pa jih s klikom na gumb Cancel (slov. prekliči) preklical. Prek spletnega vmesnika je bila uporaba orodja LanguageTool zelo preprosta, saj je bila že njegova zunanja zgradba zelo enostavna. Uporabnik je besedilo vnesel v polje in kliknil gumb Check Text (slov. preveri besedilo). Po preverjanju je pregledovalnik izpisal domnevne napake na zaslon, poleg njih pa še njihovo razlago in predlog popravka. Glavna razlika med pregledovalnikom v spletnem vmesniku in v KWordu je bila, da v spletnem vmesniku ni bilo omogočeno sprotno popravljanje.

Pregledovalnik je v ozadju v obeh programih deloval enako. Najprej je ločevalnik povedi s pomočjo regularnih izrazov in seznama kratic postavil meje med povedmi in besedami, nato je označevalnik dodelil besedam oblikoskladenjske oznake. Drobilec povedi je s pomočjo pravil razdelil na besedne zveze, s čimer je lahko pregledovalnik zaznal prve možne slovnične napake. Pravila so napisana v dveh različnih jezikih, enostavnejša so napisana

v označevalnem jeziku XML, zahtevnejša pa v jeziku Java. Po končanem preverjanju pravil je pregledovalnik vse domnevne napake zbral v seznam napak v zapisu XML, ki ga je grafični vmesnik izpisal na zaslon.

3.1.2 Zadnja različica

Prvotno različico je Naber leta 2005 objavil v prenovljeni obliki, napisano v programskem jeziku java (LanguageTool Changelog) [6]. V najnovejši različici programa LanguageTool (2.9) je podprtih 31 jezikov, med katerimi je od leta 2007 tudi slovenščina. To različico orodja LanguageTool lahko uporabljamo kot samostojen program ali kot programski dodatek k odprtokodnima pisarniškima paketoma LibreOffice in OpenOffice. Možno ga je uporabiti tudi kot razširitev za brskalnik Mozilla Firefox ali na spletni strani `https://www.languagetool.org/`, kjer so vse oblike pregledovalnika tudi prosto dostopne.

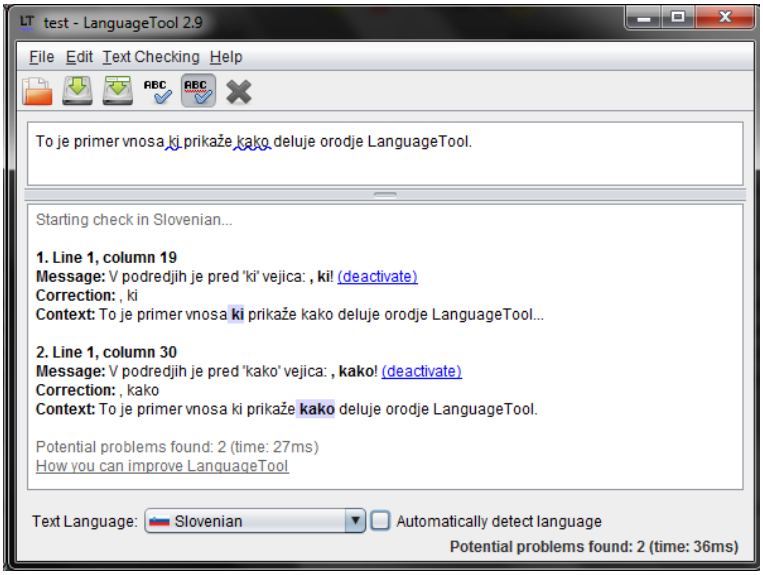
Ker je delovanje orodja LanguageTool v ozadju (ang. backend) v vseh oblikah enako, namen diplomskega dela pa je podpora analizi slovenskih besedil v okolju LanguageTool, je za nas pomembnejši ozadnji del kot uporabniški vmesnik. V nadaljevanju predstavimo delovanje samostojne namizne oblike programa, saj na njem testiramo naša pravila.

Zgradba samostojne namizne oblike programa

Zgradbo samostojnega programa za namizje delno vidimo na sliki 3.1. Razdeljena je na dva dela, v zgornjem je polje za vnos besedila, ki ga želimo preveriti, v spodnjem delu pa je polje, v katerem se izpišejo rezultati preverjanja tega besedila.

V glavnem meniju nam tretja možnost preverjanje besedila (ang. Text Checking) omogoča:

- Check Text (slov. Preglej besedilo) – pregleda besedilo v zgornjem polju in vrne rezultate.



Slika 3.1: : Glavno okno najnovejše različice samostojnega programa LanguageTool.

- AutoCheck (slov. Samodejno preglej) – če je pred to možnostjo kljukica, se pregledovanje besedila izvaja sproti ob pisanju, če to možnost izklopimo, moramo za pregled besedila klikniti možnost Check Text.
- Check Text in Clipboard (slov. Preglej besedilo v odložišču) – sprazni polje za pregledovanje besedila in vanj prilepi besedilo, ki je bilo nazadnje shranjeno v odložišče in ga pregleda.
- Tag Text (slov. Označi besedilo) – odpre okno, v katerem je besedilo oblikoskladenjsko označeno, kot to naredi Označevalnik.
- Load Rule File... (slov. Naloži datoteko s pravili) – omogoča nalaganje datoteke s pravili, ki mora biti napisana v pravilnem formatu (o tem bolj podrobno v nadaljevanju), na podlagi katerih pregledovalnik pregleda besedilo.
- Options... (slov. Možnosti ...) – odpre okno, kjer lahko vklopimo ali izklopimo posamezna pravila, ki jih uporablja pregledovalnik za pregled besedila. Določimo lahko svoj materni jezik. Omogoča nam tudi

možnost uporabe preko strežnika na izbranem vhodu (ang. port) in izberemo lahko, da se prej izbrane nastavitve uporabijo tudi na strežniku.

Na dnu glavnega okna lahko izberemo jezik, v katerem je napisano besedilo, lahko tudi odkljukamo možnost, da program samodejno zazna jezik vnesenega besedila (ang. Automatically detect language).

»Ob prenovi programa, ki je sedaj napisan v programskem jeziku Java, se njegove notranje komponente niso bistveno spreminjale, kar velja tudi za način notranjega delovanja programa.« [6]

Delovanje samostojnega namiznega programa

V tem razdelku spoznamo potek delovanja samostojnega namiznega programa.

Pregledovanje besedila poteka tako, da uporabnik besedilo vnese ali prilepi v zgornje polje v glavnem oknu, lahko odpre želeno datoteko z možnostjo Open ali pa z možnostjo Check Text in Clipboard vnese v polje za pregledovanje besedilo, ki je bilo nazadnje shranjeno v odložišče. Na koncu v primeru, da samodejno pregledovanje ni vklopljeno, pritisne gumb Check Text. Po končanem pregledu besedila program rezultate izpiše v spodnjem delu programa. Izpis rezultata vsebuje naslednje podatke: vsaka napaka ima zaporedno številko, vrstico in stolpec, kjer se je napaka pojavila. Nato je v naslednji vrstici sporočilo o napaki, ki jo je program zasledil. Poleg tega sporočila je tudi možnost izklopa tega pravila (v oklepaju na koncu sporočila je napisano deactivate (slov. izključi)), kar je mogoče storiti tudi v izbiri Text Checking pod možnostjo Options.... Zatem je vrstica, ki vsebuje popravek storjene napake. V zadnji vrstici pa je Context (slov. kontekst), ki vsebuje sobesedilo domnevne napake, ki je odebeljena in označena s svetlo modro barvo. V desnem spodnjem kotu in na koncu seznama domnevnih napak lahko zasledimo, koliko domnevnih napak je našel program.

3.2 Izboljšave orodja LanguageTool

Slovnični in slogovni pregledovalnik LanguageTool ima svojo wiki spletno stran, na kateri najdemo veliko namigov in nasvetov, kako lahko pripomoremo k izboljšanju programa. Pri tem ni pomembno, ali smo razvijalci v programskem jeziku Java, prevajalci, študenti jezikov ali prostovoljci. Med njimi najdemo tako imenovano triminutno predstavitev (ang. The three-minute introduction), kako napisati pravila za zaznavanje napak v označevalnem jeziku XML, kar potrebujemo tudi mi.

Pravila dodamo v datoteko `grammar.xml` za želeni jezik. V našem primeru želimo dodajati pravila za slovenski jezik, zato je to datoteka `LanguageTool-29/orglanguagetool/rules/sl/grammar.xml`. Po končanem dodajanju pravil, datoteko shranimo, poiščemo datoteko `languagetool.jar` in jo s klicem ukaza *java -jar languagetool.jar* v ukazni vrstici zaženemo. Pri tem se odpre samostojni namizni program LanguageTool in z vnosom testnega besedila ugotovimo, ali naše pravilo deluje.

3.2.1 Pravila

V tem razdelku opišemo sestavo napisanega pravila in kako novo napisana pravila testirati.

Osnovni elementi pravila

Pravilo je vzorec, ki v primeru ujemanja prikaže sporočilo z napako [1]. Pravilo ima 8 osnovnih elementov. Obvezne lahko vidimo na sliki 3.2, ki vsebuje primer pravila v LanguageTool 2.9. Osnovni elementi so:

- `rule` - ima dva atributa, prvi je `id`, ki mora biti unikaten in je notranji identifikator, ki se uporablja za naslavljanje pravila. Drugi atribut je `name`, ki je namenjen opisu pravila [1].

- `antipattern` - ni obvezen, gre za izjemo pravila.

```
<rulegroup name="Manjkajoča vejica pred 'toda'" id="TODA_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" negate="yes">[,\(;:\-]</token>
            <marker>
                <token>toda</token>
            </marker>
        </pattern>
        <message>Ponavadi je pred 'toda' vejica: <suggestion>, toda</suggestion>!</message>
        <short>Najbrž manjka vejica pred 'toda'</short>
        <example type="incorrect">Bilo je vroče<marker> toda</marker> nisva odnehala.</example>
        <example correction=", toda">Bilo je vroče<marker> toda</marker> nisva odnehala.</example>
    </rule>
</rulegroup>
```

Slika 3.2: Pravilo za vstavljanje manjkajoče vejice pred *toda*, ki vsebuje osnovne elemente.

- `pattern` - je podelement elementa `marker` in določa, kateri del besedila mora biti označen kot napaka.

- `token` - ima atribut `regexp`. Če je ta atribut nastavljen na `yes`, potem interpretira podano oznako `token` kot regularni izraz.

- `message` - besedilo, ki je vidno uporabniku ob zaznani napaki. Ima podelement `suggestion`, ki vsebuje sporočilo s predlagano rešitvijo napake.

- `url` - ni obvezen element. Omogoča dostop do povezave, na kateri je bolj podrobna razlaga napake.

- `short` - ni obvezen element. Omogoča prikaz kratkega opisa pravila v samostojnem namiznem programu v LibreOffice in v OpenOffice.

- `example` - najmanj dva primera z eno pravilno in eno nepravilno povedjo. Poved z atributom `type` z vrednostjo `incorrect` je primer te napake, mesto napake v povedi pa mora biti označeno s podelementom `marker`. Prvi pravilni in prvi nepravilni primer se prikažeta v samostojnem namiznem programu v podrobnostih pravila.

Testiranje pravil

Ob vsaki spremembi datoteke `grammar.xml` moramo ponovno zagnati datoteko `languagetool.jar`, saj v nasprotnem primeru program spremembe ne bo zaznal.

Pomembno je omeniti, da pravila učinkujejo zaporedno, tako da če eno pravilo z začetka datoteke `grammar.xml` že zajame vzorec kot prepoznan, program ne preverja naprej z drugim pravilom, ki šele sledi v datoteki tej datoteki, zato je včasih zelo pomembno, kam postavimo kakšno pravilo.

Za hitrejše testiranje je v LanguageTool vgrajena funkcija, ki jo v operacijskem sistemu Linux pokličemo z ukazom *sh testrules.sh sl*, v operacijskem sistemu Windows pa z ukazom *testrules.bat sl*. Ta dva ukaza testirata pravilo s povedjo iz elementa `example`. Poved, katerega atribut `type` ima vrednost `incorrect`, naj bi pravilo zaznalo in vrnilo napako. Pravilna poved ne sme vrniti napake, v nasprotnem primeru naše pravilo ali primer pravilne povedi še ni ustrezno.

3.2.2 Obstoječa pravila

Že napisana pravila so v datoteki `grammar.xml` razdeljena na šest kategorij:

- Slog - številke - pravila, da moramo števila manjša od deset in manjša od sto v besedilu pisati z besedo.
- Napake pri kraticah - pravila za pisanje nekaterih kratic.
- Napake pri postavljanju ločil - nekaj pravil za postavitev vejic pred določenimi besedami, pravilno pisanje pomišljaja, vezaja in v primeru ponavljanja ločil.
- Napake uporabe predlogov - pravila za pravilno uporabo predlogov *k, h, s* in *z*.
- Polvikanje - preverja pogoste napake pri uporabi polvikanja.

- Pogoste napake - pravila za pogoste napake, ki se pojavljajo pri pisanju npr. manjkajoči presledek med besedami, nepravilna raba besednih zvez, slogovni nasveti ...

Mi se posvečamo pravilom za napake pri postavljanju vejice pred določenimi strukturami. Nekatera izmed teh pravil pozneje nadgradimo/dopolnimo ali dodamo nova pravila, ki bodo v interakciji z obstoječimi izboljšala preverjanje besedil.

V nadaljevanju predstavimo vsa obstoječa pravila za stavo vejice. Za boljšo preglednost jih razdelimo v več skupin in podrobnosti o vsakem vezniku predstavimo v tabelah. Če podrobneje pogledamo tabele, vidimo, da je pri nekaterih veznikih med besedami, ki preklicujejo pravilo, tudi beseda, za katero kasneje vidimo, da pri naslednjem pravilu uveljavlja stavo vejice pred veznikom v kombinaciji s to besedo. To je zato, da ne dobivamo opozoril za vejico znotraj večbesednega veznika ali kombinacije predloga z veznikom.

Vejica pred prirednimi vezniki

Obstoječa pravila za manjkajočo vejico pred prirednimi vezniki so napisana za veznika *toda* in *ampak*, ki določata protivno priredje, *saj* in *kajti*, ki določata vzročno priredje, *to je* in njegova okrajšava *tj.*, ki določata pojasnjevalno priredje, *zato* in *zatorej*, ki določata sklepalno ali posledično razmerje. V razpredelnici 3.1 lahko najdemo besede, ki lahko stojijo pred ali za izbranim veznikom in s tem preklicujejo rabo pravila za stavo vejice pred tem veznikom.

Vejica pred stopnjevalnimi vezniki

Za stavo vejice pred stopnjevalnimi vezniki sta napisani dve pravili, to sta pravilo za stavo vejice pred *ampak tudi*, ki je drugi del stopnjevalnega veznika *ne le/samo - ampak tudi* in pravilo za stavo vejice pred *temveč tudi*, ki je drugi del stopnjevalnega veznika *ne le/samo - temveč tudi*. Pri teh dveh pravilih ni določenih besed, ki bi to pravilo razveljavile.

Tabela 3.1: Priredni vezniki, pred katerimi stavimo vejico, in besede, ki lahko stojijo pred in za njimi.

Veznik	**Besede pred njimi**	**Besede za njimi**
ampak	-	tudi
kajti	-	-
saj	-	-
toda	-	-
to je	in	-
tj.	-	-
torej	in, nam, jim, ji, mu, nama, vama, njima, jima, vam, meni, mi, tebi, ti	je, bo, sem, so, smo
zato	in, ter, ali, samo, le, zgolj, predvsem, lahko, ampak, glavnem, morda	, (vejica)
zatorej	in	-

Vejica pred podrednimi vezniki

Pravila za uporabo vejice pred podrednimi vezniki, lahko najdemo za veznike, ki so našteti v tabeli 3.2. V njej lahko tudi razberemo, katere besede pred in za naštetimi vezniki preklicujejo veljavnost pravila in ga ne upoštevajo.

Za razliko od ostalih pravil za stavo vejice pred podrednimi vezniki, ki preverjajo, katere besede pravilo razveljavijo, pravilo za pisanje vejice pred veznikom *kot* preverja pojavitev besed za tem veznikom, ki stavo vejice uveljavljajo, to sta veznika *če* in *da*.

Tabela 3.2: Podredni vezniki, pred katerimi stavimo vejico, in besede, ki lahko stojijo pred in za njimi.

Veznik	**Besede pred njimi**	**Besede za njimi**
ki	in, ali, ter	-

kar	že, še, v, na, pred, nad, ob, za, skozi, pod, in, ali, ter	koli, cel, precej, velik, mnogo, naprej, naenkrat, nekaj, najbolj
kaj	že, še, v, na, pred, nad, ob, za, skozi, pod, in, ali, nič, ter	-
kateri	in, ali, ter, v, na, pred, nad, pri, ob, s, za, po, iz, o, skozi, pod, izza, okoli, vzdolž, do, od, proti, med, zaradi, h	koli
kakršen	in, ali, ter, v, na, pred, nad, pri, ob, s, za, po, iz	koli
kakšen	in, ali, ter, v, na, pred, nad, pri, ob, s, za, po, iz	, (vejica)
zaradi	in, ter, ali	česar, kogar, kakršen, kateri
ko	in, ali, ter, l, samo, tudi, potem, zlasti, celo, medtem, ampak, a, vendar, toda, vsakič, takoj	-
kadar	in, ali, ter, le, samo, tudi, potem, zlasti, celo, razen	koli
kamor	in, ali, ter, le, samo, tudi, potem, zlasti, celo, razen	koli
kolikor	in, ali, ter, le, samo, tudi, potem, zlasti, celo, razen, v, na	koli
v/na kolikor	in, ali, ter, le, samo, tudi, potem, zlasti, celo, razen	koli
kakor	in, ali, ter, le, samo, tudi, potem, zlasti, celo, razen	če, bi, da, sta, smo, je, sem, so, ste, sva, se, kdo, hitro, mi, nam, ji, boš, bosta, bodo

kako	in, ali, ter, še, a	-
ker	in, ali, ter, a, temveč, ampak, vendar, toda	-
da	in, ali, ter, tako, le, samo, kot, kakor, češ, ko, lahko, temveč, ne, ampak, nam, jim, vam, mu, mi, ji, jima, vama, namesto, a	-
če	in, ali, ter, le, samo, tudi, da, zlasti, razen, ampak, a, kot	-
ne da bi	in, ali, ter	-
preden	in, ali, ter, tik, še, a	-
kje	iz, in, ali, ter	-
kjer	in, ali, ter	-
kdaj	in, ali, ter	-
kam	in, ali, ter	-
dokler	vse, in, ali, ter	-
vse dokler	in, ali, ter	-
kajne	-	-

Vejica pred podrednimi vezniki s predlogom

V orodju LanguageTool obstajajo tudi pravila za stavo manjkajoče vejice pred vezniki s predlogom, to so vezniki *kateri*, *kar* in *kaj*. V tabeli 3.3 lahko vidimo, katere besede pred in za temi vezniki pravila za stavo manjkajoče vejice razveljavljajo in kateri predlogi lahko stojijo pred njimi.

Tabela 3.3: Podredni vezniki in predlogi, ki uveljavljajo pravilo za stavo vejice pred predlogi, in besede, ki lahko stojijo pred in za njimi in s tem preklicujejo stavo manjkajoče vejice.

Vezniki	**Besede pred njimi**	**Besede za njimi**	**Predlogi**
kaj	in, ali, ter	-	v, na, pred, nad, ob, za, skozi, pod
kar	in, ali, ter	koli, nekaj, naenkrat	v, na, pred, nad, ob, za, skozi, pod
kateri	in, ali, ter, brez	koli	v, na, pred, nad, pri, ob, s, za, po, iz, o, skozi, pod, izza, okoli, vzdolž, do, proti, med, brez, od, zaradi

3.2.3 Pravila za postavljanje vejice pred veznikom *in*

Vejica pred pojasnjevalnim *in*

Implementacija tega pravila je enostavna. Kot lahko vidimo v programski kodi na slikah 3.3 in 3.4, v besedilu preverimo, ali se pojavi kateri izmed veznikov *in to* ali *in sicer*.

```
<rulegroup name="Manjkajoča vejica pred 'in sicer'" id="IN_SICER_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" negate="yes">[,\(::\-]</token>
            <marker>
                <token>in</token>
                <token>sicer</token>
            </marker>
        </pattern>
        <message>Ponavadi je pred 'in sicer' vejica: <suggestion>, in sicer</suggestion>!</message>
        <short>Najbrž manjka vejica pred 'in sicer'</short>
        <example type="incorrect">Tako lep dan je bil<marker> in sicer</marker> nisva odnehala.</example>
        <example correction=", in sicer">Tako lep dan je bil<marker> in sicer</marker> nisva odnehala.</example>
    </rule>
</rulegroup>
```

Slika 3.3: Pravilo za vstavljanje manjkajoče vejice pred *in sicer*.

```
<rulegroup name="Manjkajoča vejica pred 'in to'" id="IN_TO_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" negate="yes">[,\(::\-]</token>
            <marker>
                <token>in</token>
                <token>to</token>
            </marker>
        </pattern>
        <message>Ponavadi je pred 'in to' vejica: <suggestion>, in to</suggestion>!</message>
        <short>Najbrž manjka vejica pred 'in to'</short>
        <example type="incorrect">Predava dvakrat na teden<marker> in to</marker> v torek in petek.</example>
        <example correction=", in to">Predava dvakrat na teden<marker> in to</marker> v torek in petek.</example>
    </rule>
</rulegroup>
```

Slika 3.4: Pravilo za vstavljanje manjkajoče vejice pred *in to*.

Vejica pred vezalnim *in*

Najpogostejša pojavitev vejice pred vezalnim *in* je, ko je pred njim odvisni stavek. To sicer niso vedno odvisniki, saj lahko gre tudi za pristavke ali dostavke, vendar je te načeloma težje predvideti, tako da se osredotočimo predvsem na odvisnike. Ker so tudi odvisniki lahko zelo raznorodni, se osredotočimo predvsem na prilastkove odvisnike, in sicer poskušamo v pravilu (slika 3.5) predvideti vse veznike in vezniške strukture, v katerih se pojavljajo prilastkovi odvisniki.

```
<rulegroup name="Manjkajoča vejica pred vezalnim 'in'" id="VEZALNI_IN_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" skip="-1">ko|ki|da|če|kje</token>
            <marker>
                <token>in<exception scope="previous">,</exception></token>
            </marker>
        </pattern>
        <message>Ponavadi je pred vezalnim 'in' vejica: <suggestion>, in</suggestion>!</message>
        <short>Najbrž manjka vejica pred vezalnim 'in'</short>
        <example type="incorrect">Povej mi, kje si bil<marker> in</marker> pojdi potem na postajo.</example>
        <example correction=", in">Povej mi, kje si bil<marker>, in</marker> pojdi potem na postajo.</example>
    </rule>
</rulegroup>
```

Slika 3.5: Pravilo za vstavljanje manjkajoče vejice pred vezalnim *in*.

Problematična je prav tako sopostavitev dveh enakovrednih odvisnikov, saj mora v takem primeru orodje LanguageTool vedeti, da pred vezalnim *in* ni vejice, čeprav je pred njim oziralni odvisnik. Vejice ni zato, ker prvemu odvisniku sledi še eden, ki je prvemu enakovreden.

V prejšnjem poglavju smo povedali, da vejica stoji pred veznikom *in*, v

primeru slabe združjivosti osebkov v prirednih stavkih. Vendar je ne znamo dobro določiti, zato tega pravila ne implementiramo.

Vejica med nadrednim in odvisnim stavkom

Prvi del pravila smo implementirali že v prejšnjem primeru, kolikor se je to le dalo. Drugi del implementiramo tako, da preverimo pojavitev prirednega veznika *in* skupaj s podrednim veznikom: *in če*, *in ko*, *in ker* ipd. (slika 3.6)

```
<rulegroup name="Manjkajoča vejica pred 'in' in podrednim veznikom" id="IN_PODREDNI_VEZNIK_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" negate="yes">[,\(;:\-]</token>
            <marker>
                <token regexp="yes">in</token>
                <token regexp="yes">če|ko|ker</token>
            </marker>
        </pattern>
        <message>Ponavadi je pred 'in' in podrednim veznikom vejica: <suggestion>, in</suggestion>!</message>
        <short>Najbrž manjka vejica pred vezalnim 'in' in podrednim veznikom vejica.</short>
        <example type="incorrect">Pridem sam <marker> in če</marker> bo le mogoče, pripeljem še koga.</example>
        <example correction=", in">Pridem sam <marker>, in če</marker> bo le mogoče, pripeljem še koga.</example>
    </rule>
</rulegroup>
```

Slika 3.6: Pravilo za vstavljanje manjkajoče vejice pred *in* ter podrednim veznikom.

Vejica pred dostavkom

V tem primeru implementacija pravila ni mogoča, saj takega dostavka ne znamo prepoznati. Obstoj dostavka je odvisen od pisca, kako poudari stavek. To bi lažje prepoznali s pomočjo tehnik semantične analize besedila, katere podrobneje opisuje tudi dr. Mitja Trampuš v svoji doktorski disertaciji [10].

3.2.4 Pravila za postavljanje vejice pred veznikom *ali*

Vejica pred protivnim *ali*

Kot smo povedali že v prejšnjem poglavju, protivni *ali* prepoznamo po tem, da dopolnjevalni stavek, ki se začne z *ali*, nasprotuje osnovnemu stavku. Tudi tukaj bi bila za implementacijo potrebna semantična analiza, vendar

tega pravila ne potrebujemo implementirati, saj se protivni *ali* v sodobni jezikovni rabi ne uporablja.

Vejica pred ločnim *ali*

Veznik *ali* v ločnem priredju ločuje skladenjsko enakovredne dele. To pomeni, da imamo dve enakovredni enoti ali več, ki jih ne ločimo z vejico. Do težav lahko pride le takrat, ko imamo pred ločnim veznikom *ali* drugostopenjski stavek (npr. odvisnik), tako da mora pred tem veznikom stati vejica. V določenih primerih lahko pride do težav tudi zaradi tega, ker poznamo tri različne skladenjske vloge veznika *ali*, dve priredni in eno podredno, v dveh primerih vejico pred njim pišemo, v enem primeru pa ne.

Problem je, ko imamo dva zaporedna veznika *ali*: *Ne vem, ali bo prišel ali ne.* V tem primeru je vejica samo pred prvim, ki je podredni, in lahko rečemo, da je v zaporedju dveh veznikov *ali* vejica pred drugim. Tako tudi implementiramo to pravilo, ki ga lahko vidimo na sliki 3.7.

```
<rulegroup name="Manjkajoča vejica pred dvema zaporednima 'ali'" id="PODREDNI_ALI_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <marker>
                <token regexp="yes" skip="-1">ali<exception scope="previous">,</exception></token>
            </marker>
            <token>ali</token>
        </pattern>
        <message>Ponavadi je pred podrednim 'ali' vejica: <suggestion>, ali</suggestion>!</message>
        <short>Najbrž manjka vejica pred podrednim 'ali' vejica.</short>
        <example type="incorrect">Ne vem <marker> ali</marker> jutri pride ali ne.</example>
        <example correction=", ali">Ne vem <marker>, ali</marker> jutri pride ali ne.</example>
    </rule>
</rulegroup>
```

Slika 3.7: Pravilo za vstavljanje manjkajoče vejice pred podrednim *ali*.

Vejica pred naštevalnim *ali*

Pri implementaciji tega pravila, bi bilo pomembno le to, da preštejemo pojavitev besede *ali* v povedi in če je večja od tri, postavimo vejico pred vsako besedo *ali* v povedi. Vendar na tak primer naletimo zelo redko, zato implementacija ni potrebna.

Vejica med nadrednim in odvisnim stavkom

V orodju LanguageTool lahko odvisnik prepoznamo samo po besednem redu. Če bi uporabili označevalnik, bi podrednemu vezniku *ali* sledil bodisi glagol bodisi povratnoosebni zaimek.

Označevalnik za slovenski jezik še ni implementiran v LanguageTool. Ker bi za implementacijo le tega porabili preveč časa, ga v sklopu tega diplomskega dela ne implementiramo in ga zato posledično ne moremo uporabiti. Lahko pa se predvidijo vse pretekle in prihodnje oblike ter oblike s povratnim osebnim zaimkom, kot to lahko vidimo na sliki 3.8. Pravilo za pisanje vejice pred odvisnim *ali* ne velja, ko je pred njim veznik *in*.

```
<rulegroup name="Manjkajoča vejica pred odvisnim 'ali'" id="ODVISNI_ALI_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" negate="yes">in</token>
            <marker>
                <token>ali<exception scope="previous">,</exception></token>
            </marker>
            <token regexp="yes">se|bo|je|bi</token>
        </pattern>
        <message>Ponavadi je pred odvisnim 'ali' vejica: <suggestion>, ali</suggestion>!</message>
        <short>Mogoče manjka vejica pred odvisnim 'ali'</short>
        <example type="incorrect">Strah<marker> ali</marker> se mi stvar posreči, mi je jemal spanec.</example>
        <example correction=", ali">Strah<marker>, ali</marker> se mi stvar posreči, mi je jemal spanec</example>
    </rule>
</rulegroup>
```

Slika 3.8: Pravilo za vstavljanje manjkajoče vejice pred odvisnim *ali*.

3.2.5 Pravila za postavljanje vejice pred veznikom *da*

Vejica med nadrednim in odvisnim stavkom

Pravilo, ko veznik *da* stoji med nadrednim in odvisnim stavkom, je že bilo implementirano. Kot smo povedali v prejšnjem poglavju, je nekaj primerov, ko vejice pred veznikom *da* ni. Nekaj takih primerov je že dodanih v obstoječem pravilu, tako da se pravilo v primeru teh besed ne upošteva, nekaj izjem dodamo še mi, kar lahko vidimo na sliki 3.9.

Pravilo, ko se beseda *da* pojavi v vlogi členka, ko vejice pred njim ne pišemo, prepoznamo, ko se ta pojavi za besedami: *baje, menda, on, ona,*

```
<rulegroup name="Manjkajoča vejica pred 'da'" id="DA_BREZ_VEJICE">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes" negate="yes">[,„&quot;]|[Ii]n|[Aa]li|[Tt]er|[Tt]ako|
            [Ll]e|[Ss]amo|[Kk]ot|[Kk]akor|[Čč]eš|[Kk]o|[Ll]ahko|[Tt]emveč|[\(;:\-]|
            [Nn]e|[Aa]mpak|nam|jim|vam|mu|mi|ji|jima|vama|[Nn]amesto|[Aa]|[Bb]aje|
            [Mm]enda|on|ona|oni|[Tt]oliko|temu|tem</token>
            <marker>
                <token>da</token>
            </marker>
        </pattern>
        <message>V podredjih je pred 'da' vejica: <suggestion>, da</suggestion>!</message>
        <short>Manjka vejica pred 'da'</short>
        <example correction=", da">Mislil je<marker> da</marker> je konec sveta.</example>
        <example type="incorrect">Mislil je<marker> da</marker> je konec sveta.</example>
    </rule>
</rulegroup>
```

Slika 3.9: Pravilo za vstavljanje manjkajoče vejice pred veznikom *da*.

oni ipd. Se pravi pri implementaciji pravila samo preverimo pojavitev prej naštetih besed pred veznikom *da*.

Ravno tako je enostavna tudi implementacija pravila, ko je veznik *da* del večbesednega veznika: *namesto da, kljub temu da, češ da, tako da, toliko da, s tem da* ipd., saj je način implementacije enak kot pri prejšnjem pravilu.

Ker je glede na rezultate analize korpusa Lektor pri vezniku *da* večja težava odvečna vejica kot manjkajoča, napišemo še pravilo za opozorilo ob stavi odvečne vejice pred veznikom *da*. Ker večbesednih veznikov ni enostavno vključiti v programsko kodo, napišemo dve pravili za odvečno stavo vejice, prvo (slika 3.10) za večbesedna veznika *kljub temu da* in *s tem da* posebej in drugo (slika 3.11) pravilo za ostale primere besed pred veznikom *da*.

Kot smo že prej omenili, obstajajo tudi primeri, ko lahko del prvotnega veznika postavimo v nadredni stavek. V takem primeru prepustimo odločitev piscu, ali se bo odločil za prej napisano pravilo ali ne. Seveda bi pisca lahko na to pravilo vsaj opozorili in bi se sam odločil, ali bo pravilo upošteval ali ne, vendar bi lahko bilo to z uporabniškega vidika moteče.

```
<rulegroup name="Odvečna vejica v večbesednih veznikih 'kljub temu da'" id="KLJUB_TEMU_DA_ODVECNA_VEJICA">
    <rule>
        <pattern case_sensitive="yes">
            <patern>
                <token regexp="yes">[Kk]ljub|[Ss]</token>
                <token regexp="yes">temu|tem</token>
            </patern>
            <marker>
                <token regexp="yes">[,]</token>
                <token>da</token>
            </marker>
        </pattern>
        <message>
            Pred 'da' v večbesednih veznikih 'kljub temu da' in 's tem da' ni vejice:
            <suggestion>kljub temu da</suggestion>!
        </message>
        <short>Odvečna vejica pred 'da'</short>
        <example correction="kljub temu da">Kljub temu<marker> da</marker> je bil bolan,je šel na delo.</example>
        <example type="incorrect">Kljub temu<marker>, da</marker> je bil bolan,je šel na delo.</example>
    </rule>
</rulegroup>
```

Slika 3.10: Pravilo za stavo odvečne vejice pred veznikom v večbesednih veznikih *kljub temu da* in *s tem da*.

```
<rulegroup name="Odvečna vejica pred 'da'" id="DA_ODVECNA_VEJICA">
    <rule>
        <pattern case_sensitive="yes">
            <token regexp="yes">[Ii]n|[Aa]li|[Tt]er|[Tt]ako|Ll]e|[Ss]amo|[Kk]ot|[Kk]akor|
            [Čč]eš|[Kk]o|[Ll]ahko|[Tt]emveč|[Nn]e|[Aa]mpak|[Nn]amesto|[Aa]|[Bb]aje|
            [Mm]enda|[Oo]ziroma|[Tt]oliko</token>
            <marker>
                <token regexp="yes">[,]</token>
                <token>da</token>
            </marker>
        </pattern>
        <message>V večbesednih veznikih ponavadi pred 'da' ni vejice: <suggestion> da</suggestion>!</message>
        <short>Morda odvečna vejica pred 'da'</short>
        <example correction=" da">Namesto<marker> da</marker> bi se učil, je lenaril.</example>
        <example type="incorrect">Namesto<marker>, da</marker> bi se učil, je lenaril.</example>
    </rule>
</rulegroup>
```

Slika 3.11: Pravilo za stavo odvečne vejice med določenimi besedami in veznikom *da*.

Vejica med pastavki in drugim besedilom

Ko je beseda *da* kot samostojni členek na koncu povedi, je pravilo enako kot smo ga predstavili že v prejšnjem primeru.

Ko pa je le-ta na začetku povedi, je vejica za besedo *da*, za kar potrebujemo novo pravilo. Vendar je vlogo besede *da* kot členek v povedi nemogoče določiti, saj za njo lahko stoji katera koli besedna vrsta.

Beseda *da* na začetku povedi najpogosteje ni v samostojni členek v primeru, ko uvaja odvisnik. Najpogosteje bi to bil namerni odvisnik, ker se

ta najpogosteje rabi na začetku povedi. Zato bi bilo najbolje predvideti vse pogojne oblike neposredno za veznikom *da*. V nasprotnih primerih bi lahko LanguageTool uporabniku priporočil uporabo vejice za členkom *da*. Vendar se lahko prepogosto zgodi, da bi nas pravilo opozorilo na manjkajočo vejico, ko to ne bi bilo potrebno, zato pravila ne implementiramo.

Poglavje 4

Evalvacija pravil

Pravila med razvojem testiramo v namizni različici orodja LanguageTool. Za samo evalvacijo pravil uporabimo pisarniški paket LibreOffice s programom LibreOffice Writer.

Najprej prenesmo s spleta LibreOffice in ga namestimo na računalnik, nato prenesemo s spleta še najnovejšo različico .oxt datoteke orodja LanguageTool, katero preko možnosti v orodni vrstici `Orodja -> Upravitelj razširitev ...` dodamo v paket LibreOffice.

V operacijskem sistemu Windows poiščemo mapo `C:/Users/<uporabnik>/AppData/Roaming/LibreOffice/4/user/uno_packages/cache/uno_packages/<nakljucno_ime>.tmp_/LanguageTool-3.0.oxt/org/languagetool/rules/sl` in v njej datoteko `grammar.xml` zamenjamo s svojo popravljeno različico te datoteke. Ob vsaki spremembi datoteke moramo LibreOffice ponovno zagnati. Nato v orodni vrstici `Orodja -> Skladnja (LanguageTool) ... -> Prilagodi ...` omogočimo samo tista pravila, za katera želimo, da jih program upošteva/preverja.

Pri evalvaciji pravil preverjamo tri dejavnike: pravilnost, uporabnost in uporabniško izkušnjo vsakega pravila posebej.

Pravilnost pravil ocenimo s pomočjo daljšega besedila. Najprej preverimo pogostost zaznavanja napak v nelektorirani in nato še v lektorirani različici besedila. To besedilo je delo Simone Klemenčič z naslovom *Spisovnik pri-*

merjalnega jezikoslovca s podnaslovom *Navodila za izdelavo seminarskih, diplomskih in magistrskih del* [7]. Strukturirano je kot knjiga, razdeljeno je na poglavja in podpoglavja. Delo je priročnik s področja lingvistike in je namenjeno študentom, ki v slovenščini pišejo naloge iz primerjalnega/diahronega jezikoslovja. Besedilo je bilo lektorirano v plačljivem programu Microsoft Office Word, kjer je bila izbrana možnost sledenja spremembam, tako da lahko vidimo vsako spremembo v besedilu. Nelektorirano besedilo ima 134 strani, na teh je 51.827 besed in 293.114 znakov brez praznih prostorov (presledkov), postavljenih je 5.362 vejic.

Nelektorirano različico besedila uporabimo za evalvacijo naših pravil, lektoriranega pa uporabimo za preverjanje, ali je stava vejice, ki jo želi staviti naše pravilo, pravilno ali ne. Besedilo je za evalvacijo primerno, ker je dovolj dolgo, da lahko vsebuje dovolj primerov, da preverimo pravilnost naših pravil. Ugotovimo lahko, ali zazna tudi primere že pravilno postavljenih vejic in ali jih pravilno spregleda.

4.1 Pravila za postavljanje vejice pred veznikom *in*

4.1.1 Vejica pred pojasnjevalnim *in*

Pri zaznavanju manjkajoče vejice pred pojasnjevalnim *in*, to je v našem primeru pred večbesednima veznikoma *in sicer* in *in to*, je orodje LanguageTool z našim pravilom uspešno, saj v besedilu zazna vse pojavitve obeh prej naštetih primerov, ko pred njima ne stoji vejica, če pa je vejica že postavljena, potem tako pojavitev spregleda.

Za manjkajočo vejico pred večbesednim veznikom *in sicer* nas orodje LanguageTool opozori šestkrat, od tega je vsako opozorilo smiselno, le da so štiri opozorila, za katera zagotovo vemo, da so primerna, za dve opozorili pa ne moremo vedeti, saj se pojavita v delu, ki je v lektorirani različici popravljeno tako, da raba vejice ni potrebna oz. se pojavi v sklopu vaj, kjer

pravopisne napake morajo biti.

Za manjkajočo vejico pred večbesednim veznikom *in to* nas orodje LanguageTool opozori petkrat, od tega je dvakrat opozorilo smiselno, trikrat pa ne, saj je lahko *in to* tudi del navadnega vezalnega priredje, kjer vejice ne stavimo.

Kot smo ugotovili, najdemo primere, ko vejice ne stavimo pred tema dvema veznikoma, vendar so taki primeri izjeme. Takrat mora uporabnik sam ugotoviti, da pravilo ne velja in ga ignorirati.

Kljub nekaterim izjemam, ko naše pravilo ne velja, je pravilo uporabno in z uporabniškega vidika ni moteče.

4.1.2 Vejica pred vezalnim *in*

Pri evalvaciji tega pravila ugotovimo, da ni dovolj natančno. Pri evalvaciji smo že na prvih enajstih straneh naleteli na 18 pojavitev opozoril za to pravilo, vendar je bilo le eno opozorilo upravičeno. Iz tega lahko sklepamo, da trenutno napisano pravilo za stavo vejice pred vezalnim *in* zazna preveč primerov pojavitve besede *in*, ko le ta ne stoji za odvisnim stavkom oz. ko pred njim ne sme stati vejica.

Tako pri tem pravilu pridemo do zaključka, da ga je potrebno še izpopolniti, da bo njegova uporabnost na stopnji, da ga lahko ponudimo uporabniku. Posledično je z uporabniškega vidika moteč, saj se opozorila za to pravilo, ko to ne velja, pojavijo prevečkrat.

4.1.3 Vejica med nadrednim in odvisnim stavkom

Pri evalvaciji pravila za postavljanje vejic med nadrednim in odvisnim stavkom, kjer je veznik *in* skupaj s podrednim veznikom, je orodje zaznalo štiri primere manjkajoče vejice. Od tega so tri opozorila upravičena in eno opozorilo, ki smo ga morali prezreti. Glede na rezultate lahko sklepamo, da je pravilo implementirano ustrezno.

Ker je zaznanih več primerov, ko pravilo velja, je z uporabniškega vidika

pravilo uporabno. Za izredne primere mora poskrbeti uporabnik sam in se v takih primerih za pravilo ne odloči.

4.2 Pravila za postavljanje vejice pred veznikom *ali*

4.2.1 Vejica pred ločnim *ali*

Evalvacija za pravilo stave vejice pred ločnim *ali* ni bila uspešna, ker v izbranem besedilu ni dovolj takih primerov. Orodje LanguageTool vrne eno opozorilo, vendar pravilo v tem primeru ne velja in smo morali opozorilo prezreti. Iz tega ne moremo sklepati, ali je pravilo pravilno in ali je uporabno, niti ne moremo vedeti, če je moteče z uporabniškega vidika.

4.2.2 Vejica med nadrednim in odvisnim stavkom

V testnem besedilu so bile vejice pred odvisnim *ali* postavljene pravilno, zato smo naključnih sedem izbrisali in orodje LanguageTool je vrnilo vsa pravilna opozorila. Iz tega sklepamo, da pravilo za postavljanje vejice med nadrednim in odvisnim stavkom, ki se začne s veznikom *ali* deluje pravilno. Res je, da je pravilo samo delno določeno, saj ga ne moremo implementirati tako kot je potrebno, vendar implementirani del deluje brez težav. To pravilo je uporabno, saj z njim lahko popravimo vse pojavitve. Ker pri tem pravilu ni opaziti težav, je tudi z uporabniškega vidika koristno in nemoteče.

4.3 Pravila za postavljanje vejice pred veznikom *da*

4.3.1 Vejica med nadrednim in odvisnim stavkom

Pravilo za stavo vejice pred odvisnim veznikom *da* je že prej delovalo brezhibno in je uporabniku zelo koristilo in tudi sedaj, ko smo dodali še nekaj besed, deluje pravilno. Pravilo je vrnilo sedem napak, vendar so vsa opozorila napačna, saj zazna besedo na začetku alineje ali pa kot del da/ne odgovora na zastavljena vprašanja, ji jih je potrebno obkrožiti. Po podrobnejši analizi besedila ugotovimo, da pravilo drugih napak ne more odkriti, saj so vse vejice v besedilu pred besedo *da* pravilno postavljene. Po rezultatih evalvacije sklepamo, da je pravilo pravilno in uporabno, in z uporabniškega vidika nemoteče.

Pravilo, ki smo ga napisali za odvečno vejico pred veznikom *da*, v nelektoriranem besedilu najde 22 napak, vendar so vse napake za večbesedni veznik *tako da* in vsa opozorila se glede na lektorirano besedilo izkažejo za nepotrebna. Zato pri tem nastane težava, saj glede na pravilo, ki smo ga našli v SP 2001 [12], tam ne bi smelo biti vejice. V tem primeru dvomimo v pravilnost in uporabnost napisanega pravila, kar nas privede do razmišljanja o tem, ali bi bilo za uporabnika manj moteče, če preverjanja za veznik *tako da* sploh ne bi bilo.

Tudi drugo pravilo za odvečno vejico nas postavi v dvom, saj tako kot v prejšnjem primeru v SP 2001 [12] najdemo razlago, da pri vezniku *s tem da* pred *da* ni vejice, vendar tudi v tem primeru vseh osem zaznanih opozoril glede na lektorirano pravilo ne velja. Drugega dela pravila za veznik *kljub temu da* nismo mogli preveriti, ker besedilo ne vsebuje takega primera. Tudi tukaj podvomimo v uporabnost pravila.

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo želeli v orodju LanguageTool izpopolniti pravila za stavo vejice. Glede na analizo korpusa Lektor smo ugotovili, da stava vejice v slovenskem jeziku ni vedno enostavna in samoumevna, še posebej zato, ker so pravila skladenjska, pogosto pa upoštevajo tudi sotvarje. Nekatera pravila v SP 2001 [12] so izrazito kompleksna, razlaga pravil pa v celoti temelji na zmožnosti skladenjske analize.

S pomočjo analize korpusa Lektor in že obstoječih pravil v orodju LanguageTool smo izbrali veznike, za katere smo lahko ugotovili, da pišočim v slovenskem jeziku povzročajo težave, in se odločili, da poskusimo implementirati pravila za stavo vejice v orodju LanguageTool. Ti vezniki so: *in, ali, da*. Poglobili smo se v pravila v SP 2001 [12]. Na podlagi le-teh smo poskusili napisati pravila v obliki XML, ki bi jih lahko uporabili v orodju LanguageTool, ki je že vsebovalo nekaj enostavnejših pravil, med njimi tudi za stavo vejice pred nekaterimi vezniškimi strukturami. Uspelo nam je implementirati večino (pomembnejših) pravil, ki smo jih želeli pripraviti in jih na koncu ovrednotili.

Ugotovili smo, da je implementacija nekaterih pravil trivialna, vendar je večina težje izvedljiva. Nekatera pravila, ki smo jih uspeli implementirati, so zelo koristna, uporabna in primerna za naslednjo verzijo orodja LanguageTool, druga pa potrebujejo še nekaj popravkov.

Z našimi pravili smo izpopolnili orodje LanguageTool, zaradi preverjanja več napak omogočili piscem v slovenskem jeziku bolj brezskrbno pisanje in olajšali delo bodočim razvijalcem, ki bodo želeli napisati nova pravila.

Pri diplomskem delu smo naleteli na nekaj težav. Pravila za stavo vejice v Slovenskem pravopisu 2001 [12] niso razdeljena glede na besede, ampak glede na strukturo povedi. Pri vsakem razdelku posebej smo morali pogledati, ali je pravilo za stavo vejice napisano tudi za naše veznike ali ne. Tako smo našli tudi nekaj pravil, za katere smo kasneje ugotovili, da za nas niso relevantna in jih odstranili.

Pri implementaciji smo ugotovili, da nekaterih pravil zaradi različnih razlogov ni mogoče ali ni smiselno implementirati. Med njimi je pravilo za stavo vejice pred veznikom *in*, ko je le-ta v začetku dostavka, saj je stava vejice odvisna le od pisca, kako poudari stavek. Do implementacije ni prišlo tudi pri pravilu za vejico pred protivnim *ali* in naštevalnim *ali*, ker se v sodobnem jeziku ne pojavljata več oz. zelo redko. Zadnje pravilo, ki smo si ga zadali, a ga nismo implementirali, je pravilo za stavo vejice za besedo *da*, ki ima vlogo členka in stoji na začetku povedi, saj tudi tega ni enostavno ugotoviti.

Ostala pravila nam je večinoma uspelo implementirati. Delo so nam otežila predvsem pravila, kjer je zelo pomembna stavčna analiza. Taka pravila smo lahko implementirali le delno, saj orodje LanguageTool v trenutni različici ne vsebuje označevalnika za slovenski jezik, katerega potrebujemo za stavčno analizo.

Tako pravilo je pravilo za stavo vejice pred vezalnim *in*, ki smo le delno implementirali, ker smo lahko določili le nekaj besed, po katerih lahko poskusimo določiti, ali vejica pred *in* manjka ali ne. Zato posledično pri evalvaciji ugotovimo, da pravilo ni dovolj natančno implementirano za uporabo. Podoben primer je tudi pravilo za stavo vejice pred ločnim *ali* in tudi odvisnim *ali*, pri katerih tudi potrebujemo stavčno analizo za določitev, ali je stava vejice potrebna ali ne. Vendar za razliko od pravila za vezalni *in*, implementirana dela teh dveh pravil vseeno delujeta dovolj dobro, da ju lahko uporabljamo,

še posebej pravilo za odvisni *ali*.

S težavami smo se srečali tudi pri pisanju pravil za odvečno vejico pred veznikom *da*, saj v SP 2001 [12] najdemo primere, ko naj ne bi bilo vejice pred veznikom, vendar se pri evalvaciji izkaže nasprotno.

Za udejanjenje še dodatnih pravil potrebujemo označevalnik za slovenski jezik. Ta nam bi omogočil in poenostavil natančneje pisanje kompleksnejših pravil tako za stavo vejice kot druga pravopisna pravila. Do vključitve označevalnika nas loči korpus z dovolj veliko količino podatkov in z natančno postavljeno vejico. Seveda označevalnik ne bi rešil vseh težav, ki smo jih imeli, vendar bi jih bilo zagotovo veliko manj.

Literatura

[1] Development overview. LanguageTool Wiki. `http://wiki.languagetool.org/development-overview`. [Dostop: 25.04.2015].

[2] Korpus Lektor. `http://www.korpus-lektor.net`. [Dostop: 21.04.2015].

[3] Tomaž Erjavec. Računalniške zbirke besedil. *Jezik in slovstvo*, 1997.

[4] Damjan Popič in Darja Fišer. Vejica je mrtva, živela vejica. *Obdobja*, 2015.

[5] Damjan Popič in Nataša Logar. Med dvema ognjema: kje stoji vejica v slovenskih gimnazijah? *Obdobja*, 2015.

[6] Mario Jurišić. *Slovnični pregledovalniki za slovenščino, magistrsko delo.* Filozofska fakulteta, Univerza v Ljubljani, 2013. `http://jurisicm.webs.com/magistrska_jurisic.pdf`. [Dostop: 25.04.2015].

[7] Simona Klemenčič. *Spisovnik primerjalnega jezikoslovca. Navodila za izdelavo seminarskih, diplomskih in magistrskih del.* Narodna in univerzitetna knjižnica, Ljubljana, 2012.

[8] Daniel Naber. *A Rule-Based Style and Grammar Checker, diplomsko delo.* Technische Fakultät, Universität Bielefeld, 2003. `http://www.danielnaber.de/languagetool/download/style_and_grammar_checker.pdf`. [Dostop: 25.04.2015].

[9] Damjan Popič. *Korpusnojezikoslovna analiza vplivov na slovenska prevodna besedila, doktorska dizertacija.* Filozofska fakulteta, Univerza v Ljubljani, 2014.

[10] Mitja Trampuš. *Semantični pristopi h konstrukciji domenskih predlog in odkrivanju mnenj iz naravnega besedila, doktorska disertacija.* Fakulteta za računalništvo in informatiko, Univerza v Ljubljani, 2015.

[11] Inštitut za slovenski jezik Frana Ramovša ZRC SAZU. Slovar slovenskega knjižnega jezika. `http://bos.zrc-sazu.si/sskj.html`. [Dostop: 19.05.2015].

[12] Sodelavci Inštituta za slovenski jezik Frana Ramovša ZRC SAZU. *Slovenski pravopis 2001 Pravila.* Inštitut za slovenski jezik Frana Ramovša ZRC SAZU, 2010.